第 3 章：数据结构与算法　　作者：希赛教育软考学院　来源：希赛网　2014年05月19日

线性表

3.2 线性表

在线性表方面，主要考查栈和队列的基本操作，循环链表和双向链表的操作，二维数组的存储，以及广义表的基本概念。严格地说，广义表不是线性表，但为了方便，我们把它归结到这个知识点。

线性表是最简单和最常用的一种数据结构，线性表是由相同类型的结点组成的有限序列。一个由n个结点$a_0,a_1,\ldots,a_{n-1}$组成的线性表可记为（$a_0,a_1,\ldots,a_{n-1}$）。线性表的结点个数为线性表的长度，长度为0的线性表称为空表。对于非空线性表，a_0是线性表的第一个结点，a_{n-1}是线性表的最后一个结点。线性表的结点构成一个序列，对序列中两相邻结点a_i和a_{i+1}称a_i是a_{i+1}的前驱结点，a_{i+1}是a_i的后继结点。其中a_0没有前驱结点，a_{n-1}没有后继结点。

线性表中结点之间的关系可由结点在线性表中位置所确定，通常用（a_i,a_{i+1}）（$0\le i\le n-2$）表示两个结点之间的先后关系。例如：如果两个线性表有相同的数据结点，但它们的结点在线性表中出现的顺序不同，则它们是两个不同的线性表。

线性表的结点可由若干个成分组成，其中能唯一标识该结点的成分称为关键字，或简称键。为了讨论方便，往往只考虑结点的关键字，而忽略其他成分。

上一节　本书简介　下一节

第 3 章：数据结构与算法　　作者：希赛教育软考学院　来源：希赛网　2014年05月19日

栈

3.2.1 栈

栈是一种特殊的线性表，栈只允许在同一端进行插入和删除运算。允许插入和删除的一端称为栈顶，另一端称为栈底。称栈的结点插入为进栈，结点删除为出栈。因为最后进栈的结点必定最先出栈，所以栈具有后进先出（先进后出）的特征。

1.顺序存储

可以用顺序存储线性表来表示栈，为了指明当前执行插入和删除运算的栈顶位置，需要一个地址变量top指出栈顶结点在数组中的下标。

2.链接存储栈

栈也可以用链表实现，用链表实现的栈称为链接栈。链表的第一个结点为顶结点，链表的首结点就是栈顶指针top,top为NULL的链表是空栈。

队列

3.2.2 队列

队列也是一种特殊的线性表，只允许在一端进行插入，另一端进行删除运算。允许删除运算的那一端称为队首，允许插入运算的那一端称为队尾。称队列的结点插入为进队，结点删除为出队。因为最先进入队列的结点将最先出队，所以队列具有先进先出的特征。

1.顺序存储

可以用顺序存储线性表来表示队列，为了指明当前执行出队运算的队首位置，需要一个指针变量head（称为头指针）。为了指明当前执行进队运算的队尾位置，也需要一个指针变量tail（称为尾指针）。

若用有N个元素的数组表示队列，随着一系列进队和出队运算，队列的结点移向存放队列数组的尾端，会出现数组的前端空着，而队列空间已用完的情况。一种可行的解决办法是当发生这样的情况时，把队列中的结点移到数组的前端，修改头指针和尾指针。另一种更好的解决办法是采用循环队列。

循环队列就是将实现队列的数组a[N]的第一个元素a[0]与最后一个元素a[N−1]连接起来（本章以C语言的规则表示，下同）。一般情况下，用tail指向队尾元素的下一个位置（注意：并不是指向最后一个元素本身），用head指向队头元素。队空的初态为head=tail=0.在循环队列中，当tail赶上head时，队列满。反之，当head赶上tail时，队列变为空。这样队空和队满的条件都同为head=tail,这会给程序判别队空或队满带来不便。因此，可采用当队列只剩下一个空闲结点的空间时，就认为队列已满，用这种简单办法来区别队空和队满。即队空的判别条件是head=tail,队满的判别条件是head=（tail+1）%N.

2.链接存储

队列也可以用链接存储线性表实现，用链表实现的队列称为链接队列。链表的第一个结点是队列首结点，链表的末尾结点是队列的队尾结点，队尾结点的链接指针值为NULL.队列的头指针head指向链表的首结点，队列的尾指针tail指向链表的尾结点。当队列的头指针head值为NULL时，队列为空。

链表

3.2.3 链表

链表就是采用链式存储实现的线性表。根据其存储结构的不同，可以分为单链表、循环链表和双向链表三种。

1.单链表

单链表的每个结点包括两个域，分别是数据域和指针域next（指向下一个结点），最后一个结点的指针为NULL,head表示头指针。如图3-1所示。

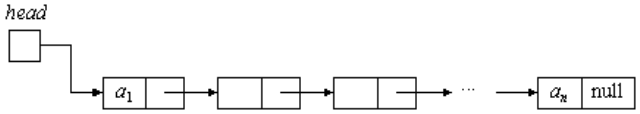


图3-1 单链表

在单链表中删除一个结点q的过程如图3-2所示。

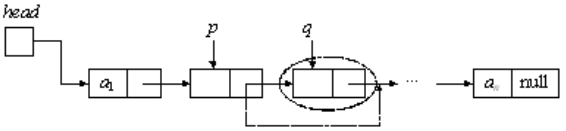


图3-2 在单链表中删除结点

（1）找到要删除的结点q和该结点的前趋p.

（2）将p的指针指向q所指向的后继结点：pànext=qànext.

该操作的算法平均时间复杂度为O（n）。

在单链表中插入一个结点x的过程如图3-3所示。

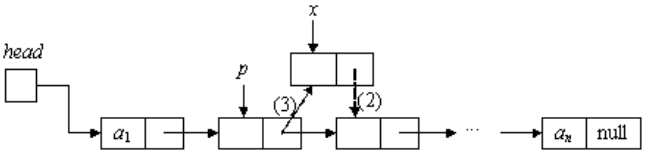


图3-3 在单链表中插入结点

（1）找到插入点的前趋p.

（2）将要插入的结点x的指针指向p的后继结点：xànext=pànext.

（3）将p的指针指向x:pànext=x.

该操作的算法平均时间复杂度为O（n）。

要注意：步骤2和3不能互换，否则将产生信息的丢失。

2.循环链表

循环链表与单链表的区别仅仅在于其尾结点的指针域值不是NULL,而是指向头结点的指针。这样做的好处是从表中的任一结点出发都能够通过后移操作扫描整个循环链表。循环链表的例子如图3-4所示。

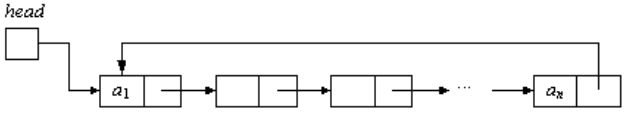


图3-4 循环链表的例子

3.双向链表

双向链表（双链表）的每个结点包括三个域，分别是数据域、前趋指针prior（指向前一个结点）和后继指针next（指向后一个结点），最后一个结点的后继指针为null.如图3-5所示。

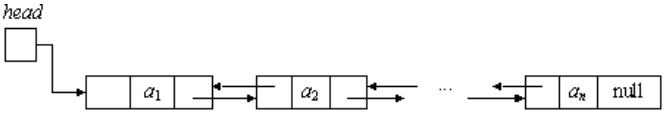


图3-5 双链表

在双链表中删除一个结点p的过程如图3-6所示。

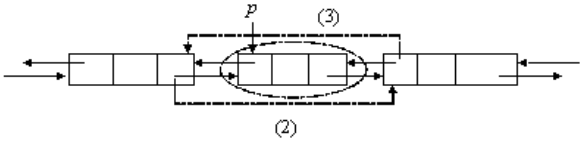


图3-6 在双链表中删除一个结点

（1）找到要删除的结点p.

（2）将p的前趋结点的后继指针指向p的后继结点：pàpriorànext=pànext.

（3）将p的后继结点的前趋指针指向p的前趋结点：pànextàprior=pàprior.

在双链表中插入一个结点x的过程如图3-7所示。

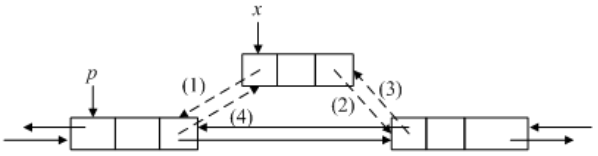


图3-7 在双链表中插入一个结点

（1）将待插入的结点x的前趋指针指向插入点p:xàprior=p.

（2）将插入的结点x的后继指针指向插入点p的后继结点：xànext=pànext.

（3）将p的后继结点的前趋指针指向x:pànextàprior=x.

（4）将p的后继指针指向x:pànext=x.

上一节　本书简介　下一节

二维数组

3.2.4 二维数组

数组可以看作是一种线性表，采用的是一种顺序存储方式，因此当需要插入一个新元素时，就需要通过移动原有元素挪出位置。要在a[i]之前插入新数，则需要将a[i]开始移动，由于i<N,因此后面N-i个数也需要顺序向后移动一个位置。

根据数组下标的个数，可以把数组分为一维、二维、……多维数组。维度是指下标的个数。一维数组只有一个下标；二维数组则有两个下标，第一个称为行下标，第二个称为列下标。根据数组的定义，计算存储地址的方法如表3-3所示。

表3-3 数据的存储地址计算公式

数组类型	存储地址计算
一维数组 $a[n]$，n 为个数	对于 $a[i]$的存储地址：$a+i\times len$ （a 为数组首地址，len 为每个数据对象的长度）
二维数组 $a[m][n]$，m、n 为个数	对于 $a[i][j]$的存储地址：$a+(i\times n+j)\times len$ （a 为数组首地址，len 为每个数据对象的长度）

即对于二维数组a[m][n]而言（m和n都是从0开始计数），a[i][j]的存储地址应该是a+（i×n+j）×len,如果以行为主序，则i表示行号、j表示列号；如果以列为主序，则i表示列号、j表示行号，len是每个元素的长度。

例如：对于二维数组a[04,15],设每个元素占1个存储单元，且以列为主序存储，则元素a[2,2]相对于数组空间起始地址的偏移量是0+（1×5+2）×1=7.（要注意，这里的行号是从0开始计数的，

列号是从1开始计数的。所以，第2列就相当于表3-3中的i=1）

广义表

3.2.5 广义表

广义表是n（n?0）个表元素组成的有限序列，它是递归定义的线性结构，是一个多层次的线性结构，是线性表的推广，记做：

$$LS=(a_0,a_1,...a_n)$$

其中LS是表名，ai是表元素，它可以是表（称做子表），也可以是数据元素（称为原子）。其中n是广义表的长度（也就是最外层包含的元素个数），n=0的广义表为空表。而递归定义的重数就是广义表的深度，直观地说，就是定义中所含括号的重数（原子的深度为0,空表的深度为1）。

例如：广义表L=（（1,2,3））,则其长度就是1,而深度则是2.广义表L=（（1,2,3），（4,5,（6,7,8）））的长度为2,深度为3.

二叉树

3.3 二叉树

本节要掌握的主要内容有二叉树的概念和性质、二叉树遍历，以及完全二叉树、平衡二叉树、二叉排序树、哈夫曼树（最优二叉树）等一些特殊的二叉树。

二叉树是一个有限的结点集合，该集合或者为空，或者由一个根结点及其两棵互不相交的左、右二叉子树所组成。二叉树的结点中有两棵子二叉树，分别称为左子树和右子树。因为二叉树可以为空，所以二叉树中的结点可能没有子结点，也可能只有一个左子结点（右子结点），也可能同时有左右两个子结点。

与树相比，二叉树可以为空，空的二叉树没有结点（树至少有一个结点）。在二叉树中，结点的子树是有序的，分左、右两棵子二叉树。

作者：希赛教育软考学院　来源：希赛网
2014年05月19日

二叉树的性质

3.3.1 二叉树的性质

二叉树具有下列重要性质：

性质1 在二叉树的第i层上至多有2^{i-1}≥1）。

性质2 深度为k的二叉树至多有2^k-1个结点（k≥1）。

性质3 对任何一棵二叉树，如果其叶子结点数为n0,度为2的结点数为n2,则$n_0=_{n2}+1$.

一棵深度为k且有2^k-1（k≥1）个结点的二叉树称为满二叉树。如图3-8所示就是一棵满二叉树，对结点进行了顺序编号。

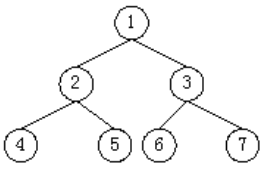


图3-8 满二叉树的例子

如果深度为k、有n个结点的二叉树中各结点能够与深度为k的顺序编号的满二叉树从1到n标号的结点相对应，则称这样的二叉树为完全二叉树。如图3-9（a）所示是一棵完全二叉树，而（b）、（c）是两棵非完全二叉树。显然，满二叉树是完全二叉树的特例。

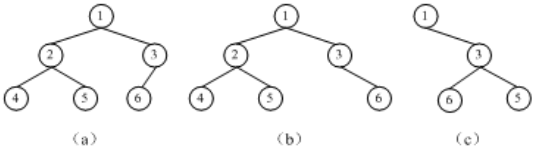


图3-9 完全二叉树和非完全二叉树

根据完全二叉树的定义，显然，在一棵完全二叉树中，所有的叶子结点都出现在第k层或k–1层（最后两层）。

性质4 具有n（n>0）个结点的完全二叉树的深度为⌊$\log_2 n$⌋+1.（注：⌊ ⌋符号为向下取整运算符，⌈ ⌉为向上取整运算符，$\lfloor m \rfloor$表示不大于m的最大整数，反之，$\lceil m \rceil$表示不小于m的最小整数）

性质5 如果对一棵有n个结点的完全二叉树的结点按层序编号（从第1层到第⌊$\log_2 n$⌋+1层，每层从左到右），则对任一结点i（1≤i≤n），有：

（1）如果i=1,则结点i无双亲，是二叉树的根；如果i>1,则其双亲是结点$\lfloor i/2 \rfloor$。

（2）如果2_i>n,则结点i为叶子结点，无左孩子；否则，其左孩子是结点2_i.

（3）如果2_i+1>n,则结点i无右孩子；否则，其右孩子是结点2_i+1.

在考试时，对这几个性质的灵活应用是十分关键的，因此应熟练的掌握它们，其实这些性质都十分明显，只需绘制出二叉树，结合着体会将能更有效地记忆。下面我们来看一个例子。

在一棵完全二叉树中，其根的序号为1,如果要判断序号为P和Q的两个结点是否在同一层，可以借助什么公式？如果这棵二叉树有767个结点，那么它有多少个叶子结点？

前一个问题的关键在于能够灵活利用完全二叉树的深度计算公式，该公式同样可以用来计算某个特定序号结点的深度，两个结点的深度如果相同，就可以说明它们在同一个层内，也就是如果"$\lfloor \log_2 P \rfloor+1=\lfloor \log_2 Q \rfloor+1$"满足，则可以说明P和Q在同一层。

后一个问题，则应该结合二叉树和完全二叉树的性质进行解答。我们用n_0表示度为0（叶子）的结点总数，用n1表示度为1的结点总数，n2为度为2的结点总数，n表示这棵完全二叉树的结点总数。显然，$n= n_0+n_1+n_2$,将性质3中的公式代入，可得到$n=2n_0+ n_1-1$.而在本题中，n=767,因此，$2_{n0}+ n1-1=767$.

由于完全二叉树中度为1的结点数只有两种可能，要么为0个（满二叉树），要么为一个。如果取值为1,则式子为2n0+1-1=767,无合理解；取值0,则式子为$2_{n0}+0-1=767$,得到解为384.

第 3 章：数据结构与算法　作者：希赛教育软考学院　来源：希赛网　2014年05月19日

二叉树的遍历

3.3.2 二叉树的遍历

二叉树的遍历方法有以下3种：

（1）前序遍历（先根遍历，先序遍历）：首先访问根结点，然后按前序遍历根结点的左子树，再按前序遍历根结点的右子树。

（2）中序遍历（中根遍历）：首先按中序遍历根结点的左子树，然后访问根结点，再按中序遍历根结点的右子树。

（3）后序遍历（后根遍历，后序遍历）：首先按后序遍历根结点的左子树，然后按后序遍历根结点的右子树，再访问根结点。

例如如图3-10所示的二叉树，其前序遍历、中序遍历和后序遍历结果分别如下。

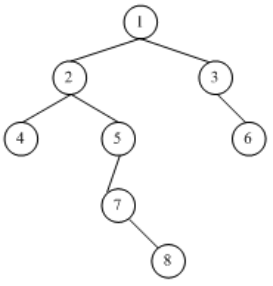


图3-10 二叉树遍历的例子

（1）前序遍历：1、2、4、5、7、8、3、6.

（2）中序遍历：4、2、7、8、5、1、3、6.

（3）后序遍历：4、8、7、5、2、6、3、1.

以上3种遍历方法都是递归定义的，可通过递归函数分别加以实现。

性质6　一棵二叉树的前序序列和中序序列可以唯一地确定这棵二叉树。

根据性质6,给定一棵二叉树的前序序列和中序序列，我们可以写出该二叉树的后序序列。例如，某二叉树的前序序列为 ABHFDECKG,中序序列为 HBDFAEKCG.因为在前序序列中A在最前面，所以A为二叉树的根。然后根据中序序列，A前面的为A的左子树，A后面的为A的右子树。对于A的左子树HBDF而言，其对应的前序序列中B在前面，因此，B为左子树的根。同理，在中序序列中，B的左边为B的左子树，B的右边为B的右子树，依此类推，就可以构造出二叉树。

有关表达式的求解问题，也与二叉树的遍历有关。例如，我们要求表达式a*（b+c）-d的后缀表达形式。解这种题，如果我们知道前缀、中缀、后缀表达式有何关联，有什么特点，那么解题就非常轻松了。其实前缀、中缀、后缀的得名，是从二叉树而来的，也就是把一个表达式转化为一棵二叉树后，对二叉树进行前序遍历得到前缀表达式，进行中序遍历得到中缀表达式（也就是一般形式的表达式），进行后序遍历得到后缀表达式。

因此，我们只要把表达式转成树的形式，再对二叉树进行后序遍历，即可得到正确答案。但现在最主要的问题是如何构造这棵树。构造的规则是这样的，所有的操作数只能在叶子结点上，操作符是它们的根结点，括号不构造到二叉树当中去，构造树的顺序要遵循运算的顺序。在表达式a*（b+c）-d中最先计算b+c,所以先构造图3-11的部分。

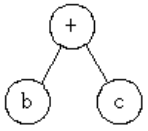


图3-11 第一步

然后把b+c的结果与a进行运算，所以有图3-12所示的结果。

接着把运算结果和d相减，所以最终得到的二叉树如图3-13所示。

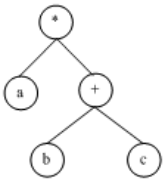


图3-12 第二步

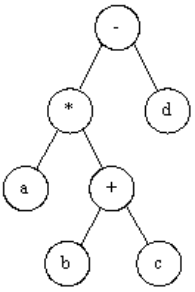


图3-13 最终得到的二叉树

对此树进行后序遍历，得到序列abc+*d-,此为表达式a*（b+c）-d的后缀表达形式。

上一节　本书简介　下一节

二叉排序树

3.3.3 二叉排序树

二叉排序树又称为二叉查找树，其定义为二叉排序树或者是一棵空二叉树，或者是具有如下性质（BST性质）的二叉树：

（1）若它的左子树非空，则左子树上所有结点的值均小于根结点。

（2）若它的右子树非空，则右子树上所有结点的值均大于根结点。

（3）左、右子树本身又各是一棵二叉排序树。

例如，如图3-14所示就是一棵二叉排序树。

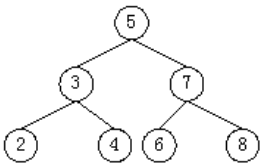


图3-14 二叉排序树的例子

根据二叉排序树的定义可知：如果中序遍历二叉排序树，就能得到一个排好序的结点序列。

第 3 章：数据结构与算法　作者：希赛教育软考学院　来源：希赛网　2014年05月19日

最优二叉树

3.3.4 最优二叉树

树的路径长度是从树根到树中每一结点的路径长度之和。在结点数目相同的二叉树中，完全二叉树的路径长度最短。在一些应用中，赋予树中结点的一个有某种意义的实数，这些数字称为结点的权。结点到树根之间的路径长度与该结点上权的乘积，称为结点的带权路径长度。树中所有叶结点的带权路径长度之和称为树的带权路径长度（树的代价），通常记为：

其中n表示叶子结点的数目，w_i和l_i分别表示叶结点k_i的权值和根到结点k_i之间的路径长度。

在权值为$w_l,w_2,\ldots$，w_n的n个叶子所构成的所有二叉树中，带权路径长度最小（即代价最小）的二叉树称为最优二叉树或哈夫曼树。

假设有n个权值，则构造出的哈夫曼树有n个叶子结点。n个权值分别设为$w_1,w_2,\ldots$，w_n,则哈夫曼树的构造规则为：

（1）将$w_1,w_2,\ldots$，w_n看成是有n棵树的森林（每棵树仅有一个结点）。

（2）在森林中选出两个根结点的权值最小的树合并，作为一棵新树的左、右子树，且新树的根结点权值为其左、右子树根结点权值之和。

（3）从森林中删除选取两棵树，并将新树加入森林。

（4）重复第（2）和（3）步，直到森林中只剩一棵树为止，该树即为所求的哈夫曼树。

利用哈夫曼树很容易求出给定字符集及其概率（或频度）分布的最优前缀码。哈夫曼编码是一种应用广泛且非常有效的数据压缩技术，该技术一般可将数据文件压缩掉20%至90%,其压缩效率取决于被压缩文件的特征。

例如：如果叶子结点的权值分别为1、2、3、4、5、6,则构造哈夫曼树的过程如图3-15所示，其带权路径长度为（1+2）×4+3×3+（4+5+6）×2=51.

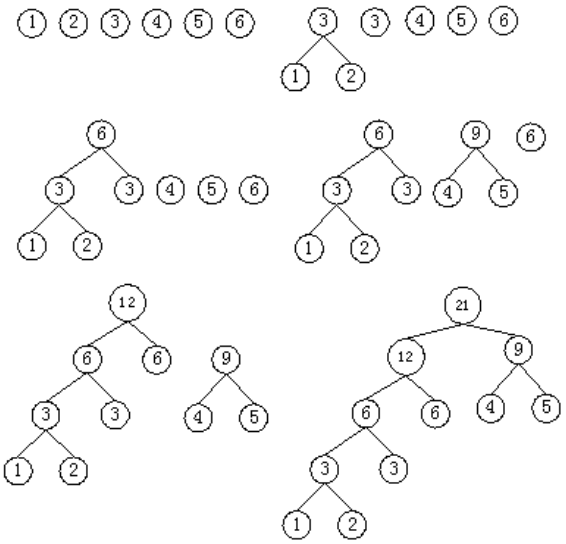


图3-15 哈夫曼树的构造过程

在构造哈夫曼树的过程中，每次都是选取两棵最小权值的二叉树进行合并，因此使用的是贪心算法。

上一节　本书简介　下一节

作者：希赛教育软考学院　来源：希赛网　2014年05月19日

平衡二叉树

3.3.5 平衡二叉树

为了保证二叉排序树的高度为$\log_2 n$,从而保证二叉排序树上实现的插入、删除和查找等基本操作的平均时间为O（$\log_2 n$），在往树中插入或删除结点时，要调整树的形态来保持树的"平衡".使之既保持BST性质不变，又保证树的高度在任何情况下均为log2n,从而确保树上的基本操作在最坏情况下的时间均为O（$\log_2 n$）。

平衡二叉树又称为AVL树，是指树中任一结点的左右子树的高度大致相同。如果任一结点的左右子树的高度均相同（如满二叉树），则二叉树是完全平衡的。通常，只要二叉树的高度为O（1$\log_2 n$），就可看做是平衡的。

AVL树中任一结点的左、右子树的高度之差的绝对值不超过1.若将二叉树上结点的平衡因子定义为该结点的左子树的深度减去它的右子树的深度，则平衡二叉树上所有结点的平衡因子只可能是–1、0和1.

在最坏情况下，n个结点的AVL树的高度约为1.44$\log_2 n$.而完全平衡的二叉树度高约为$\log_2 n$,AVL树是接近最优的。

例如：在图3-16中，假设结点A的右子树AR高度为h,结点B的左子树BL高度为h,结点C的左子树CL、右子树CR高度都为h-1.根据这些条件，我们可以得出，结点B的右子树BR（结点C为根）的高度为h.因此，结点A的左子树AL（结点B为根）子树的高度为h+1.所以，图3-16所示的二叉树为平衡二叉树。

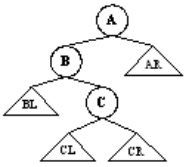


图3-16 平衡二叉树的例子

如果在CR中插入一个结点，并使得 CR 的高度增加1,则结点C的左右子树高度之差为1,同时以C为根的子树高度增加了1,所以结点B的左右子树高度之差变为1.如此一来，A的左子树的高度为h+2、而右子树的高度为h.这样，以A为根的二叉树就变为不平衡了。

上一节　本书简介　下一节

排序

3.4 排序

本节要掌握各种排序算法的基本思想，以及各种排序算法的时间比较。

所谓排序就是要整理文件中的记录，使之按关键字递增（或递减）次序排列起来。当待排序记录的关键字均不相同时，排序结果是唯一的，否则排序结果不唯一。

在待排序的文件中，若存在多个关键字相同的记录，经过排序后这些具有相同关键字的记录之间的相对次序保持不变，该排序方法是稳定的；若具有相同关键字的记录之间的相对次序发生变化，则称这种排序方法是不稳定的。

要注意的是排序算法的稳定性是针对所有输入实例而言的。即在所有可能的输入实例中，只要有一个实例使得算法不满足稳定性要求，则该排序算法就是不稳定的。

上一节　本书简介　下一节

插入排序

3.4.1 插入排序

插入排序的基本思想是每步将一个待排序的记录按其排序码值的大小，插到前面已经排好的文件中的适当位置，直到全部插入完为止。插入排序方法主要有直接插入排序和希尔排序。

1.直接插入排序

直接插入排序的过程为在插入第i个记录时，$R_1R_2,\ldots R_{i-1}$已经排好序，将第i个记录的排序码k_i依次和$R_1,R_2,\ldots$，R_{i-1}的排序码逐个进行比较，找到适当的位置。使用直接插入排序，对于具有n个记录的文件，要进行n−1趟排序。

2.希尔排序

希尔（Shell）排序的基本思想是先取一个小于n的整数d1作为第一个增量，把文件的全部记录分成d_1个组。所有距离为d_i的倍数的记录放在同一个组中。先在各组内进行直接插入排序；然后，取第二个增量$d_2<d_1$重复上述的分组和排序，直至所取的增量dt=1（$d_t<d_{t-l}<O<d_2<d_1$），即所有记录放在同一组中进行直接插入排序为止。该方法实质上是一种分组插入方法。

一般取$d_1=n/2$,$d_i+1=d_i/2$.如果结果为偶数，则加1,保证d_i为奇数。

例如：我们要对关键码{72,28,51,17,96,62,87,33,45,24}进行排序，这里n=10,首先取$d_1=n/2=5$.则排序过程如图3-17所示。

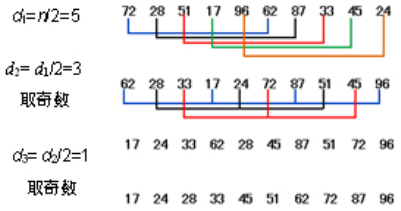


图3-17 希尔排序的过程

希尔排序是不稳定的，希尔排序的执行时间依赖于增量序列，其平均时间复杂度为O（$n^{1.3}$）。

上一节　本书简介　下一节

作者：希赛教育软考学院　来源：希赛网　2014年05月20日

选择排序

3.4.2 选择排序

选择排序的基本思想是每步从待排序的记录中选出排序码最小的记录，顺序存放在已排序的记录序列的后面，直到全部排完。选择排序中主要使用直接选择排序和堆排序。

1.直接选择排序

直接选择排序的过程是首先在所有记录中选出排序码最小的记录，把它与第1个记录交换，然后在其余的记录内选出排序码最小的记录，与第2个记录交换……依次类推，直到所有记录排完为止。

无论文件初始状态如何，在第i趟排序中选出最小关键字的记录，需做n–i次比较，因此，总的比较次数为n（n–1）/2=O（n^2）。当初始文件为正序时，移动次数为0;文件初态为反序时，每趟排序均要执行交换操作，总的移动次数取最大值3（n–1）。直接选择排序的平均时间复杂度为O（n^2）。直接选择排序是不稳定的。

2.堆排序

堆是一种特殊的完全二叉树，它采用顺序存储。如果从0开始对树的结点进行编号，编号的顺序按层进行，同层则按从左到右的次序；则编号为 $0-\lfloor n/2\rfloor-1$ 的结点为分支结点，编号大于 $\lfloor n/2\rfloor-1$ 的结点为叶结点，对于每个编号为i的分支结点，它的左子结点的编号为2_{i+1},右子结点的编号为2_{i+2}.除编号为0的树根结点外，对于每个编号为i的结点，它的父结点的编号为 $\lfloor(i-1)/2\rfloor$ 。假定结点i中存放记录的排序码为S_i,则堆的各结点的排序码满足 $S_i \cdot S_{2i+1}$ 且 $S_i \cdot S_{2i+2}$ $(0 \cdot i \cdot \lfloor n/2\rfloor-1)$。这种堆称为

大顶堆（大根堆），而如果相反（即满足$S_i \cdot S_{2i+1}$且$S_i \cdot S_{2i+2}$），则称为小顶堆（小根堆）。

若将此序列所存储的向量R[1...n]看做是一棵完全二叉树的存储结构，则堆实质上是满足如下性质的完全二叉树，即二叉树中任一非叶结点的关键字均不大于（或不小于）其左右孩子（若存在）结点的关键字。如图3-18所示的二叉树就是一个大根堆。

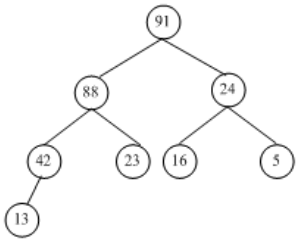


图3-18 大根堆的例子

堆排序的关键步骤有两个，一是如何建立初始堆；二是当堆的根结点与堆的最后一个结点交换后，如何对少了一个结点后的结点序列做调整，使之重新成为堆。

上一节　本书简介　下一节

第 3 章：数据结构与算法　作者：希赛教育软考学院　来源：希赛网　2014年05月20日

交换排序

3.4.3 交换排序

交换排序的基本思想是两两比较待排序记录的排序码，并交换不满足顺序要求的那些偶对，直到满足条件为止。交换排序的主要方法有冒泡排序和快速排序。

1.冒泡排序

冒泡排序将被排序的记录数组R[1...n]垂直排列，每个记录R[i]看做是重量为k_i的气泡。根据轻气泡不能在重气泡之下的原则，从下往上扫描数组R:凡扫描到违反本原则的轻气泡，就使其向上"飘浮".如此反复进行，直到最后任何两个气泡都是轻者在上，重者在下为止。

冒泡排序的具体过程如下。

第一步，先比较k_1和k_2,若$k_1 > k_2$,则交换k1和k2所在的记录，否则不交换。继续对k_2和k_3重复上述过程，直到处理完k_{n-1}和k_n.这时最大的排序码记录转到了最后位置，称第1次起泡，共执行n−1次比较。

与第一步类似，从k_1和k_2开始比较，到k_{n-2}和k_{n-1}为止，共执行n−2次比较，称第2次起泡。

依次类推，共做n−1次起泡，完成整个排序过程。

若文件的初始状态是正序的，一趟扫描即可完成排序。所需的关键字比较次数为n−1次，记录移动次数为0.因此，冒泡排序最好的时间复杂度为O（n）。

若初始文件是反序的，需要进行n−1趟排序。每趟排序要进行n−i次关键字的比较（$1 \le i \le n-1$），且每次比较都必须移动记录三次来达到交换记录位置。在这种情况下，比较次数达到最大值$n(n-1)/2=O(n^2)$，移动次数也达到最大值$3n(n-1)/2=O(n^2)$。因此，冒泡排序的最坏时间复杂度为$O(n^2)$。

虽然冒泡排序不一定要进行n−1趟，但由于它的记录移动次数较多，故平均时间性能比直接插入排序要差得多。冒泡排序是就地排序，且它是稳定的。

2.快速排序

快速排序采用了一种分治的策略，通常称其为分治法。其基本思想是将原问题分解为若干个规模更小但结构与原问题相似的子问题。递归地解这些子问题，然后将这些子问题的解组合为原问题的解。

快速排序的具体过程如下。

第一步，在待排序的n个记录中任取一个记录，以该记录的排序码为准，将所有记录分成两组，第1组各记录的排序码都小于等于该排序码，第2组各记录的排序码都大于该排序码，并把该记录排在这两组中间。

第二步，采用同样的方法，对左边的组和右边的组进行排序，直到所有记录都排到相应的位置为止。

例如：我们要对关键码{7,2,5,1,9,6,8,3}进行排序，选择第一个元素为基准。第一趟排序的过程如图3-19所示。

要注意的是在快速排序中，选定了以第一个元素为基准，接着就拿最后一个元素和第一个元素比较，如果大于第一个元素，则保持不变，再拿倒数第二个元素和基准比较，如果小于基准，则进行交换。交换之后，再从前面的元素开始与基准比较，如果小于基准，则保持不变；如果大于基准，则交换。交换之后，再从后面开始比较，依次类推，前后交叉进行。

然后，再采取同样的办法对{3,2,5,1,6}和{8,9}分别进行排序，具体过程如图3-20所示。

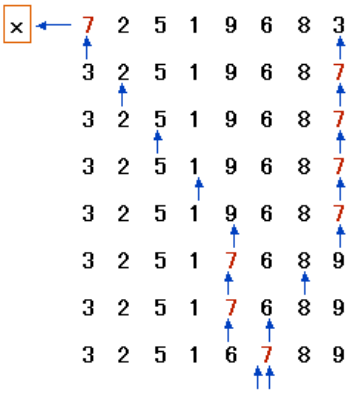


图3-19 第一趟排序过程

(1) [3 2 5 1 6] 7 [8 9]

(2) [1 2]3[5 6] 7 [8 9]

(3) 1 2 3[5 6] 7 [8 9]

(4) 1 2 3 5 6 7 [8 9]

结果 1 2 3 5 6 7 8 9

图3-20 各趟排序过程

快速排序的时间主要耗费在划分操作上，对长度为k的区间进行划分，共需k−1次关键字的比较。

最坏情况是每次划分选取的基准都是当前无序区中关键字最小（或最大）的记录，划分的结果是基准左边的子区间为空（或右边的子区间为空），而划分所得的另一个非空的子区间中记录数目，仅仅比划分前的无序区中记录个数减少一个。因此，快速排序必须做n−1次划分，第i次划分开始时区间长度为n−i+1,所需的比较次数为n−i（1≤i≤n−1），故总的比较次数达到最大值n（n−1）/2=O（n^2）。如果按上面给出的划分算法，以每次取当前无序区的第1个记录为基准，那么当文

件的记录已按递增序（或递减序）排列时，每次划分所取的基准就是当前无序区中关键字最小（或最大）的记录，则快速排序所需的比较次数反而最多。

在最好情况下，每次划分所取的基准都是当前无序区的"中值"记录，划分的结果是基准的左、右两个无序子区间的长度大致相等。总的关键字比较次数O（$_n\log_{2n}$）。

因为快速排序的记录移动次数不大于比较的次数，所以快速排序的最坏时间复杂度应为O（n^2），最好时间复杂度为O（$_n\log_{2n}$）。

尽管快速排序的最坏时间为O（n2），但就平均性能而言，它是基于关键字比较的内部排序算法中速度最快者，快速排序亦因此而得名。它的平均时间复杂度为O（$_n\log_{2n}$）。快速排序在系统内部需要一个栈来实现递归。若每次划分较为均匀，则其递归树的高度为O（$\log_{2n}$），故递归后需栈空间为O（$\log_{2n}$）。在最坏情况下，递归树的高度为O（n），所需的栈空间为O（n）。快速排序是不稳定的。

上一节　本书简介　下一节

第 3 章：数据结构与算法　作者：希赛教育软考学院　来源：希赛网　2014年05月20日

归并排序

3.4.4 归并排序

归并排序是将两个或两个以上的有序子表合并成一个新的有序表。初始时，把含有n个结点的待排序序列看做由n个长度都为1的有序子表所组成，将它们依次两两归并得到长度为2的若干有序子表，再对它们两两合并。直到得到长度为n的有序表，排序结束。

例如：我们需要对关键码{72,28,51,17,96,62,87,33}进行排序，其归并过程如图3-21所示。

归并排序是一种稳定的排序，可用顺序存储结构，也易于在链表上实现。对长度为n的文件，需进行log2n趟二路归并，每趟归并的时间为O（n），故其时间复杂度无论是在最好情况下还是在最坏情况下均是O（$_n\log_{2n}$）。归并排序需要一个辅助向量来暂存两个有序子文件归并的结果，故其辅助空间复杂度为O（n），显然它不是就地排序。

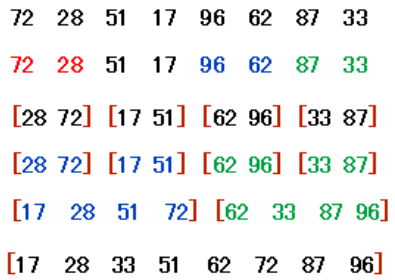


图3-21 归并排序的过程

上一节　本书简介　下一节

基数排序

3.4.5 基数排序

设单关键字的每个分量的取值范围均是$C0 \le k_j \le C_{rd-1}$（$0 \le j < rd$），可能的取值个数rd称为基数。基数的选择和关键字的分解因关键字的类型而异。

（1）若关键字是十进制整数，则按个、十等位进行分解，基数rd=10,C_0=0,C_9=9,d为最长整数的位数。

（2）若关键字是小写的英文字符串，则rd=26,C_0='a',C_{25}='z',d为字符串的最大长度。

基数排序的基本思想是从低位到高位依次对待排序的关键码进行分配和收集，经过d趟分配和收集，就可以得到一个有序序列。

基数排序的具体实现过程如下。

设有r个队列，队列的编号分别为0,1,2,…，r–1.首先按最低有效位的值把n个关键字分配到这r个队列中，然后从小到大将各队列中关键字再依次收集起来；接着再按次低有效位的值把刚刚收集起来的关键字分配到r个队列中。重复上述收集过程，直至最高有效位，这样便得到一个从小到大的有序序列。为减少记录移动的次数，队列可以采用链式存储分配，称为链式基数排序。每个队列设有两个指针，分别指向队头和队尾。

例如：我们需要对{288,371,260,531,287,235,56,299,18,23}进行排序，因为这些数据最高位为百位，所以需要分三趟分配和收集。第一趟分配和收集（按个位数）的过程如图3-22所示。第二趟分配与收集（按十位数）的过程如图3-23所示。第三趟分配与收集（按百位数）的过程如图3-24所示。

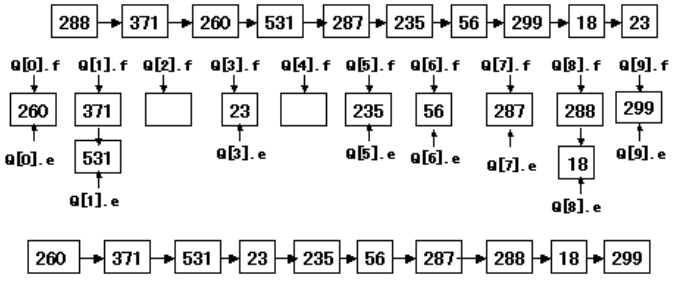


图3-22 第一趟分配与收集

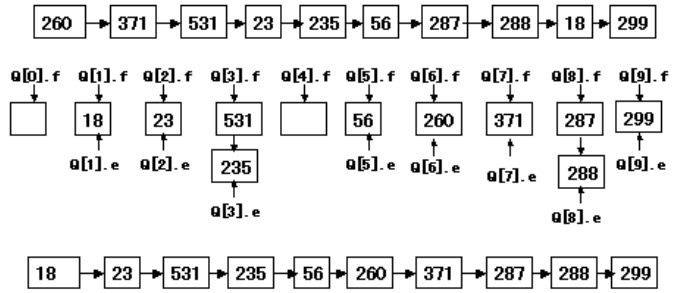


图3-23 第二趟分配与收集

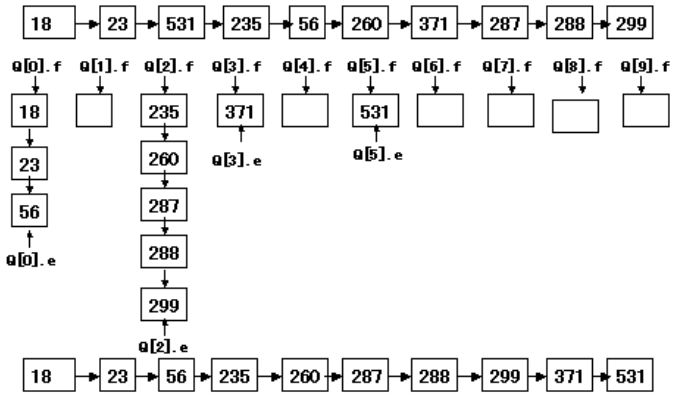


图3-24 第三趟分配与收集

基数排序的时间复杂度为O（d（r+n））,所需的辅助存储空间为O（n+rd），基数排序是稳定的。

上一节 本书简介 下一节

排序算法的比较

3.4.6 排序算法的比较

在此，我们把常用的排序算法的复杂度进行列表，如表3-4所示。

表3-4 排序算法时间复杂度表

类 别	排序方法	时间复杂度		空间复杂度	稳 定 性
		平均情况	最坏情况	辅助存储	
插入排序	直接插入	$O(n^2)$	$O(n^2)$	$O(1)$	稳定的
	Shell 排序	$O(n^{1.3})$	$O(n^2)$	$O(1)$	不稳定
选择排序	直接选择	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
	堆排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(1)$	不稳定
交换排序	冒泡排序	$O(n^2)$	$O(n^2)$	$O(1)$	稳定的
	快速排序	$O(n\log_2 n)$	$O(n^2)$	$O(n\log_2 n)$	不稳定
归并排序		$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n)$	不稳定
基数排序		$O(d(r+n))$	$O(d(r+n))$	$O(rd+n)$	稳定的

上一节 本书简介 下一节

查找

3.5 查找

在查找方面，主要考查二分法查找（折半查找）和散列法。

第 3 章：数据结构与算法　作者：希赛教育软考学院　来源：希赛网　2014年05月20日

二分法查找

3.5.1 二分法查找

二分法查找又称折半查找，它是一种效率较高的查找方法。二分法查找要求线性表是有序表，即表中结点按关键字有序，并且要用向量作为表的存储结构。

二分法查找的基本思想是（设R[low,…，high]是当前的查找区间）：

（1）确定该区间的中点位置：mid=[（low+high）/2].

（2）将待查的k值与R[mid].key比较，若相等，则查找成功并返回此位置，否则须确定新的查找区间，继续二分查找，具体方法如下：

若R[mid].key>k,则由表的有序性可知R[mid,…，n].key均大于k,因此若表中存在关键字等于k的结点，则该结点必定是在位置mid左边的子表R[low,…，mid−1]中。因此，新的查找区间是左子表R[low,…，high],其中high=mid-1.

若R[mid].key<k,则要查找的k必在mid的右子表R[mid+1,…，high]中，即新的查找区间是右子表R[low,…，high],其中low=mid+1.

若R[mid].key=k,则查找成功，算法结束。

（3）下一次查找是针对新的查找区间进行，重复步骤（1）和（2）。

（4）在查找过程中，low逐步增加，而high逐步减少。如果high<low,则查找失败，算法结束。

因此，从初始的查找区间R[1,…，n]开始，每经过一次与当前查找区间的中点位置上的结点关键字的比较，就可确定查找是否成功，不成功则当前的查找区间就缩小一半。这一过程重复直至找到关键字为K的结点，或者直至当前的查找区间为空（即查找失败）时为止。

例如：我们要在{11,13,17,23,31,36,40,47,52,58,66,73,77,82,96,99}中查找58的过程如图3-25所示（粗体表示mid位置）。在上述序列中查找35的过程如图3-26所示。

11 13 17 23 31 36 40 **47** 52 58 66 73 77 82 96 99

11 13 17 23 31 36 40 47 52 58 66 **73** 77 82 96 99

11 13 17 23 31 36 40 47 52 **58** 66 73 77 82 96 99

图3-25 二分法查找58

11 13 17 23 31 36 40 **47** 52 58 66 73 77 82 96 99

11 13 17 **23** 31 36 40 47 52 58 66 73 77 82 96 99

11 13 17 23 31 **36** 40 47 52 58 66 73 77 82 96 99

11 13 17 23 **31** 36 40 47 52 58 66 73 77 82 96 99

图3-26 二分法查找35

二分法查找过程可用二叉树来描述：把当前查找区间的中间位置上的结点作为根，左子表和右子表中的结点分别作为根的左子树和右子树。由此得到的二叉树，称为描述二分查找的判定树或比较树。要注意的是判定树的形态只与表结点个数n相关，而与输入实例中R[1,…，n].key的取值无关。

在等概率假设下，二分法查找成功时的平均查找长度约为$\log_2$（n+1）−1.二分法查找在查找失

败时所需比较的关键字个数不超过判定树的深度，在最坏情况下查找成功的比较次数也不超过判定树的深度。即为$[\log_2 n]+1$.

二分法查找只适用顺序存储结构。为保持表的有序性，在顺序结构里插入和删除都必须移动大量的结点。因此，二分法查找特别适用于那种一经建立就很少改动，而又经常需要查找的线性表。对那些查找少而又经常需要改动的线性表，可采用链表作为存储结构，进行顺序查找。链表上无法实现二分法查找。

上一节　本书简介　下一节

散列表

3.5.2 散列表

散列表，又称杂凑表，是一种十分实用的查找技术，具有极高的查找效率。其基本思想是根据关键码值（查找码）与表项存储位置的映射关系，进行高效、精确的查找。查找效率与散列函数的计算复杂度、冲突解决方案的使用有直接的关系。散列查找是根据关键码（查找码）与表项存储位置的映射关系，进行高效、精确的查找，因此也需要采用顺序结构存储。

当关键码比较多时，很可能出现散列函数值相同的情况，即出现冲突。我们通常也称为冲突的两个关键是该散列函数的同义词。通常解决冲突的方法是设法在散列表中找一个空位。而常见的方法有两种：开放定址法和拉链法。

（1）开放定址法：当冲突发生时，使用某种探查技术在散列中形成一个探序列。沿着该序列查找，直到找到关键字或一个开放的地址（地址单元为空）。

（2）拉链法：将散列表的每个结点增加一个指针字段，用于链接同义词的子表，链表中的结点都是同义词。

上一节　本书简介　下一节

图

3.6 图

本知识点重点在于掌握图的基本概念和存储，以及求关键路径、最短路径、拓扑排序的方法。

上一节　本书简介　下一节

图的基本概念

3.6.1 图的基本概念

图G由两个集合V和E组成，记为G=（V,E），其中V是顶点的有穷非空集合，E是V中顶点偶对（称为边）的有穷集合。通常，也将图G的顶点集和边集分别记为V（G）和E（G）。E（G）可以是空集。若E（G）为空，则图G只有顶点而没有边。

图分为有向图和无向图两种。图3-27（a）是一个有向图，在有向图中，一条有向边是由两个顶点组成的有序对，有序对通常用尖括号表示。$<V_i,V_j>$表示一条有向边，Vi是边的始点（起点），Vj是边的终点，$<V_i,V_j>$和$<V_j,V_i>$是两条不同的有向边。例如：在图3-27（a）中，$<V_1,V_4>$和$<V_4,V_1>$是两条不同的边。有向边也称为弧，边的始点称为弧尾，终点称为弧头。

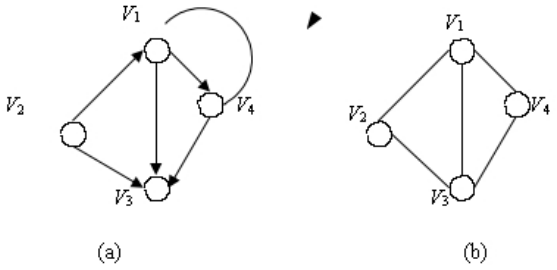


图3-27 图的分类

图3-27（b）是一个无向图，无向图中的边均是顶点的无序对，无序对通常用圆括号表示。在无向图G中，如果i≠j,i、j∈V,（i,j）∈E,即i和j是G的两个不同的顶点，（i,j）是G中一条边，顶点i和顶点j是相邻顶点，边（i,j）是与顶点i和j相关联的边。

如果限定任何一条边或弧的两个顶点都不相同，则有n个顶点的无向图至多有n（n –1）/2条边，这样的无向图称为无向完全图。一个有向图至多有n（n–1）条弧，这样的有向图称为有向完全图。

在无向图中，一个顶点的度等于与其相邻接的顶点个数。在有向图中，一个顶点的入度等于邻接到该顶点的顶点个数，其出度等于邻接于该顶点的个数。

在图G=（V,E）中，如果存在顶点序列（$V_0,V_1,\ldots,V_k$）其中$V_0=P,V_k=Q$,且（V_0,V_1），（V_1,V_2），…，（V_{k-1},V_k）都在E中，则称顶点P到顶点Q有一条路径，并用（V_0,V1,…，V_k）表示这条路径，路径的长度是路径的边数，这条路径的长度为k.若G是有向图，则路径也是有向的。

在有向图G中，若对于V（G）中任意两个不同的顶点V_i和V_j,都存在从V_i到V_j及从V_j到V_i的路径，则称G是强连通图。

有向图的极大强连通子图称为G的强连通分量。强连通图只有一个强连通分量，即是其自身。非强连通的有向图有多个强连分量。

图的存储结构

3.6.2 图的存储结构

最常用的图的存储结构有邻接矩阵和邻接表。

1.邻接矩阵

邻接矩阵反映顶点间的邻接关系，设G=（V,E）是具有n（n≥1）个顶点的图，G的邻接矩阵M是一个n行n列的矩阵，并有若（i,j）或<i,j>∈E,则M[i][j]=1;否则，M[i][j]=0.例如：图3-27（a）和（b）的邻接矩阵分别如下。

$$M_a=\begin{pmatrix}0&0&1&1\\1&0&1&0\\0&0&0&0\\1&0&1&0\end{pmatrix}\qquad M_b=\begin{pmatrix}0&1&1&1\\1&0&1&0\\1&1&0&1\\1&0&1&0\end{pmatrix}$$

由邻接矩阵的定义可知，无向图的邻接矩阵是对称的，有向图的邻接矩阵不一定对称。对于无向图，其邻接矩阵第i行元素的和即为顶点i的度。对于有向图，其邻接矩阵的第i行元素之和为顶点i的出度，而邻接矩阵的第j列元素之和为顶点j的入度。

若将图的每条边都赋上一个权，则称这种带权图为网（络）。如果图G=（V,E）是一个网，若（i,j）或<i,j>∈E,则邻接矩阵中的元素M[i][j]为（i,j）或<i,j>上的权。若（i,j）或<i,j>$\notin$E,则M[i][j]为无穷大或为大于图中任何权值的实数。

2.邻接表

在图的邻接表中，为图的每个顶点建立一个链表，且第i个链表中的结点代表与顶点i相关联的一条边或由顶点i出发的一条弧。有n个顶点的图，需用n个链表表示，这n个链表的头指针通常由顺序线性表存储。例如：图3-27（a）和（b）的邻接表分别如图3-28（a）和（b）所示。

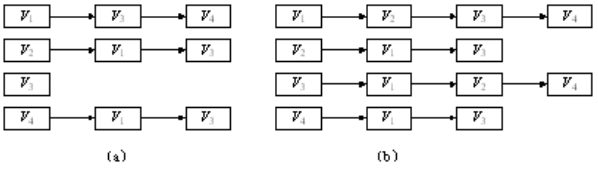


图3-28 图的邻接表表示

在无向图的邻接表中，对应某结点链表的结点个数就是该顶点的度。在有向图的邻接表中，对应某结点链表的结点个数就是该顶点的出度。

上一节　本书简介　下一节

第 3 章：数据结构与算法　作者：希赛教育软考学院　来源：希赛网　2014年05月20日

图的遍历

3.6.3 图的遍历

图的遍历是从某个顶点出发，沿着某条搜索路径对图中每个顶点各做一次且仅做一次访问。它是许多图的算法的基础。深度优先遍历和广度优先遍历是最为重要的两种遍历图的方法。它们对无

向图和有向图均适用。

1.深度优先遍历

在G中任选一顶点V为初始出发点（源点），则深度优先遍历可定义为首先访问出发点V,并将其标记为已访问过；然后依次从V出发搜索V的每个邻接点W.若W未曾访问过，则以W为新的出发点继续进行深度优先遍历，直至图中所有和源点V有路径相通的顶点（亦称为从源点可达的顶点）均已被访问为止。若此时图中仍有未访问的顶点，则另选一个尚未访问的顶点作为新的源点重复上述过程，直至图中所有顶点均已被访问为止。

图的深度优先遍历类似于树的前序遍历。对于无向图，如果图是连通的，那么按深度优先遍历时，可遍历全部顶点，得到全部顶点的一个遍历序列。如果遍历序列没有包含所有顶点，那么该图是不连通的。

2.广度优先遍历

广度优先的遍历过程是首先访问出发顶点V,然后访问与顶点V邻接的全部未被访问过的顶点W_0,W_1,…，W_{k-1};接着再依次访问与顶点W_0,W_1,…，W_{k-1}邻接的全部未被访问过的顶点。依次类推，直至图的所有顶点都被访问到，或出发顶点V所在的连通分量的全部顶点都被访问到为止。

从广度优先搜索遍历过程可知，若顶点V在顶点W之前被访问，则对V相邻的顶点的访问就先于只与W相邻的那些顶点的访问。因此，需要一个队列来存放被访问过的顶点，以便按顶点的访问顺序依次访问这些顶点相邻接的其他还未被访问过的顶点。

第 3 章：数据结构与算法　作者：希赛教育软考学院　来源：希赛网　2014年05月20日

拓扑排序

3.6.4 拓扑排序

对一个有向无环图G进行拓扑排序，是将G中所有顶点排成一个线性序列，使得图中任意一对顶点u和v,若<u,v> ∈E（G），则u在线性序列中出现在v之前。这样的线性序列称为满足拓扑次序的序列，简称拓扑序列。要注意的是：

（1）若将图中顶点按拓扑次序排成一行，则图中所有的有向边均是从左指向右的。

（2）若图中存在有向环，则不可能使顶点满足拓扑次序。

（3）一个有向无环图可能有多个拓扑序列。

（4）当有向图中存在有向环时，拓扑序列不存在。

一个大工程有许多项目组，有些项目的实行则存在先后关系，某些项目必须在其他一些项目完成之后才能开始实行。工程项目实行的先后关系可以用一个有向图来表示，工程的项目称为活动，有向图的顶点表示活动，有向边表示活动之间开始的先后关系。这种有向图表示的活动网络，简称AOV网络，如图3-29所示。

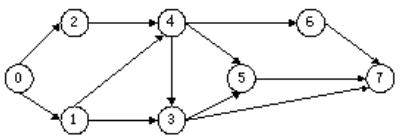


图3-29 AOV网络的例子

对AOV网络的顶点进行拓扑排序，就是对全部活动排成一个拓扑序列，使得如在AOV网络中存在一条弧（i,j），则活动i排在活动j之前。例如：对图3-29的有向图的顶点进行拓扑排序，可以得到多个不同的拓扑序列，如02143567,02143657,02143657,01243567等。

第 3 章：数据结构与算法 作者：希赛教育软考学院 来源：希赛网 2014年05月20日

最短路径

3.6.5 最短路径

带权图的最短路径问题即求两个顶点间长度最短的路径。其中路径长度不是指路径上边数的总和，而是指路径上各边的权值总和。路径长度的的具体含义取决于边上权值所代表的意义。

已知有向带权图（简称有向网）G=（V,E），找出从某个源点s∈V到V中其余各顶点的最短路径，称为单源最短路径。

目前，求单源最短路径主要使用迪杰斯特拉（Dijkstra）提出的一种按路径长度递增序列产生各顶点最短路径的算法。若按长度递增的次序生成从源点s到其他顶点的最短路径，则当前正在生成的最短路径上除终点以外，其余顶点的最短路径均已生成（将源点的最短路径看作是已生成的源点到其自身的长度为0的路径）。

迪杰斯特拉算法的基本思想是设S为最短距离已确定的顶点集（看作红点集），V-S是最短距离尚未确定的顶点集（看作蓝点集）。

（1）初始化：初始化时，只有源点s的最短距离是已知的（SD（s）=0），故红点集S={s},蓝点集为空。

（2）重复以下工作，按路径长度递增次序产生各顶点最短路径：在当前蓝点集中选择一个最短距离最小的蓝点来扩充红点集，以保证算法按路径长度递增的次序产生各顶点的最短路径。当蓝点集中仅剩下最短距离为∞的蓝点，或者所有蓝点已扩充到红点集时，s到所有顶点的最短路径就求出来了。

需要注意的是：

（1）若从源点到蓝点的路径不存在，则可假设该蓝点的最短路径是一条长度为无穷大的虚拟路径。

（2）从源点s到终点v的最短路径简称为v的最短路径；s到v的最短路径长度简称为v的最短距离，并记为SD（v）。

根据按长度递增序产生最短路径的思想，当前最短距离最小的蓝点k的最短路径：

源点，红点1,红点2,...，红点n,蓝点k

距离：源点到红点n最短距离 + <红点n,蓝点k>的边长

为求解方便，可设置一个向量D[0...n-1],对于每个蓝点v∈V-S,用D[v]记录从源点s到达v且除v外中间不经过任何蓝点（若有中间点，则必为红点）的"最短"路径长度（简称估计距离）。若k是蓝点集中估计距离最小的顶点，则k的估计距离就是最短距离，即若D[k]=Min{D[i] i∈V-S},则

D[k]=SD（k）。

初始时，每个蓝点v的D[c]值应为权w<s,v>,且从s到v的路径上没有中间点，因为该路径仅含一条边<s,v>.

将k扩充到红点后，剩余蓝点集的估计距离可能由于增加了新红点k而减小，此时必须调整相应蓝点的估计距离。对于任意的蓝点j,若k由蓝变红后使D[j]变小，则必定是由于存在一条从s到j且包含新红点k的更短路径：P=<s,…，k,j>.且D[j]减小的新路径P只可能是由于路径<s,…，k>和边<k,j>组成。所以，当lenGth（P）=D[k]+w<k,j>小于D[j]时，应该用P的长度来修改D[j]的值。

例如：我们求图3-30所示的图从s点到t点的最短路径。

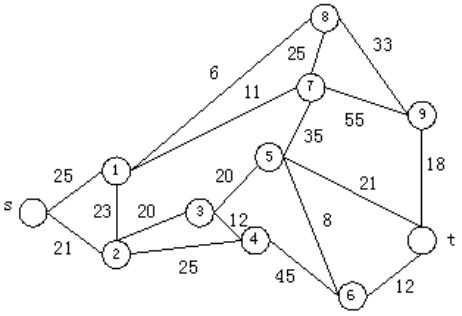


图3-30 对结点进行编号

求最短路径的过程如表3-5所示。

表3-5 求最短路径的过程

红点集	D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]	D[8]	D[9]	D[t]
{s}	25	21	∞	∞	∞	∞	∞	∞	∞	∞
{s,2}	25		41	46	∞	∞	∞	∞	∞	∞
{s,2,1}			41	46	∞	∞	36	31	∞	∞
{s,2,1,8}			41	46	∞	∞	36		64	∞
{s,2,1,8,7}			41	46	71	∞			64	∞
{s,2,1,8,7,3}				46	61	∞			64	∞
{s,2,1,8,7,3,4}					61	91			64	∞
{s,2,1,8,7,3,4,5}						69			64	82
{s,2,1,8,7,3,4,5,9}						69				82
{s,2,1,8,7,3,4,5,9,6}										81
{s,2,1,8,7,3,4,5,9,6,t}										

因此，从s到t的最短路径长度为81,路径为s→2→3→5→6→t。

上一节　本书简介　下一节

关键路径

3.6.6 关键路径

在AOV网络中，如果边上的权表示完成该活动所需的时间，则称这样的AOV为AOE网络。例如：图3-31表示一个具有10个活动的某个工程的AOE网络。图中有7个结点，分别表示事件1~7,其中1表示工程开始状态，7表示工程结束状态，边上的权表示完成该活动所需的时间。

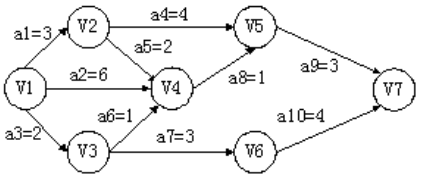


图3-31 AOE网络的例子

因AOE网络中的某些活动可以并行进行，所以完成工程的最少时间是从开始结点到结束结点的最长路径长度，称从开始结点到结束结点的最长路径为关健路径（临界路径），关键路径上的活动为关键活动。

为了找出给定的AOE网络的关键活动，从而找出关键路径，先定义几个重要的量：

$$\begin{cases} V_e(1)=0 \\ V_e(j)=MAX\{V_e(i)+d(i,j)\}, <V_i,\ V_j>\in E_1, 2\le j\le n \end{cases}$$

$V_e(j)$、$V_l(j)$：结点j事件最早、最迟发生时间。

e（i）、l（i）：活动i最早、最迟开始时间。

从源点V1到某结点V_j的最长路径长度，称为事件V_j的最早发生时间，记作$V_e(j)$。$V_e(j)$也是以V_j为起点的出边$<V_j,V_k>$所表示的活动a_i的最早开始时间e（i）。

在不推迟整个工程完成的前提下，一个事件V_j允许的最迟发生时间，记作$V_l(j)$。显然，l（i）=$V_l(j)$-（a_i所需时间），其中j为a_i活动的终点。满足条件l（i）=e（i）的活动为关键活动。

求顶点V_j的$V_e(j)$和$V_l(j)$可按以下两步来做：

$$\begin{cases} V_l(n)=V_e(n) \\ V_l(j)=MIN\{V_l(k)-d(j,k)\}, <V_j,\ V_k>\in E_2, 2\le j\le n-1 \end{cases}$$

（1）由源点开始向汇点递推

其中，E_1是网络中以V_j为终点的入边集合。

（2）由汇点开始向源点递推

其中，E_2是网络中以V_j为起点的出边集合。

要求一个AOE的关键路径，一般需要根据以上变量列出一张表格，逐个检查。例如：求图3-31所示的AOE的关键路径的表格如表3-6所示。

表3-6 求关键路径的过程

V_j	$V_e(j)$	$V_l(j)$	a_i	$e(i)$	$l(i)$	$l(i)-e(i)$
V_1	0	0	$a_1(3)$	0	0	0
V_2	3	3	$a_2(6)$	0	0	0
V_3	2	3	$a_3(2)$	0	1	1
V_4	6	6	$a_4(4)$	3	3	0
V_5	7	7	$a_5(2)$	3	4	1
V_6	5	6	$a_6(1)$	2	5	3
V_7	10	10	$a_7(3)$	2	3	1
			$a_8(1)$	6	6	0
			$a_9(3)$	7	7	0
			$a_{10}(4)$	5	6	1

因此，图3-31的关键活动为a_1,a_2,a_4,a_8和a_9,其对应的关键路径有两条，分别为（V_1,V_2,V_5,V_7）和（V_1,V_4,V_5,V_7），长度都是10.

在实际解答试题时，一般所给出的活动数并不多，我们可以采取观察法求得其关键路径，即路径最长的那条路径就是关键路径。

常用算法设计

3.7 常用算法设计

根据考试大纲，算法设计是每年必考的知识点，一般以C语言的形式进行描述。本章主要介绍经常考的几种算法的基本思想，有关这些算法的实现和实例，请读者学习《希赛软件设计师视频教程》。

算法设计概述

3.7.1 算法设计概述

算法是在有限步骤内求解某一问题所使用的一组定义明确的规则，一个算法应该具有以下5个重要的特征。

（1）有穷性：一个算法必须总是（对任何合法的输入值）在执行有穷步之后结束，且每一步都可在有穷时间内完成。

（2）确定性：算法中每一条指令必须有确切的含义，读者理解时不会产生二义性。在任何条件下，算法只有唯一的一条执行路径，即对于相同的输入只能得出相同的输出。

（3）输入：一个算法有0个或多个的输入，以刻画运算对象的初始情况。所谓0个输入是指算法本身定出了初始条件。这些输入取自于某个特定对象的集合。

（4）输出：一个算法有一个或多个输出，以反映对输入数据加工后的结果。没有输出的算法是毫无意义的。

（5）可行性：一个算法是可行的，即算法中描述的操作都是可以通过已经实现的基本运算执行有限次来实现的。

算法设计要求正确性、可读性、健壮性、效率与低存储量。效率指的是算法执行时间。对于解决同一问题的多个算法，执行时间短的算法效率高。存储量需求指算法执行过程中所需要的最大存储空间。两者都与问题的规模有关。

算法的复杂性是算法效率的度量，是算法运行所需要的计算机资源的量，是评价算法优劣的重要依据。可以从一个算法的时间复杂度与空间复杂度来评价算法的优劣。当我们将一个算法转换成程序并在计算机上执行时，其运行所需要的时间取决于下列因素：

（1）硬件的速度。

（2）书写程序的语言。实现语言的级别越高，其执行效率就越低。

（3）编译程序所生成目标代码的质量。对于代码优化较好的编译程序其所生成的程序质量较高。

（4）问题的规模。

显然，在各种因素都不能确定的情况下，很难比较出算法的执行时间。也就是说，使用执行算法的绝对时间来衡量算法的效率是不合适的。为此，可以将上述各种与计算机相关的软、硬件因素都确定下来，这样一个特定算法的运行工作量的大小就只依赖于问题的规模（通常用正整数n表示），或者说它是问题规模的函数。

1.时间复杂度

一个程序的时间复杂度是指程序运行从开始到结束所需要的时间。

一个算法是由控制结构和原操作构成的，其执行时间取决于两者的综合效果。为了便于比较同一问题的不同的算法，通常的做法是：从算法中选取一种对于所研究的问题来说是基本运算的原操作，以该操作重复执行的次数作为算法的时间度量。一般情况下，算法中原操作重复执行的次数是规模n的某个函数T（n）。

2.空间复杂度

一个程序的空间复杂度是指程序运行从开始到结束所需的存储量。

程序运行所需的存储空间包括以下两部分。

（1）固定部分：这部分空间与所处理数据的大小和个数无关。主要包括程序代码、常量、简单变量、定长成分的结构变量所占的空间。

（2）可变部分：这部分空间大小与算法在某次执行中处理的特定数据的大小和规模有关。例如：100个数据元素的排序算法与1000个数据元素的排序算法所需的存储空间显然是不同的。

算法由数据结构来体现，所以看一个程序首先要搞懂程序实现所使用的数据结构，如解决装箱问题就使用链表这种数据结构。数据结构是算法的基础，数据结构支持算法，如果数据结构是递归的，算法也可以用递归来实现，如二叉树的遍历。

第 3 章：数据结构与算法　作者：希赛教育软考学院　来源：希赛网　2014年05月20日

迭代法

3.7.2 迭代法

迭代法适用于方程（或方程组）的求解，是使用间接方法求方程近似根的一种常用算法。它的主要思想是从某个点出发，通过某种方式求出下一个点，使得其离要求的点（方程的解）更近一步；当两者之差接近到可接受的精度范围时，就认为找到了问题的解。由于它是不断进行这样的过程，因此称为迭代法，同时从中也可以看出使用迭代法必须保证其收敛性。

在使用迭代法的过程中，应该注意两种异常情况：

（1）如果方程无解，那么近似根序列将不会收敛，迭代过程会成为死循环。因此在使用时应先判断其是否有解，并应对迭代的次数进行限制，以防死循环。

（2）当方程虽然有解，但迭代公式选择不当，或迭代的初始近似根选择不合理，也会导致迭代

失败。

穷举搜索法

3.7.3 穷举搜索法

穷举搜索法是穷举所有可能的情形，并从中找出符合要求的解，即对可能是解的众多候选解按某种顺序逐一枚举和检验，并从中找出那些符合要求的解作为问题的解。

这种方法对没有有效解法的离散型问题，规模不大时，穷举法不失是一种可考虑的方法。但规模较大时，穷举显然就不行了。

穷举搜索法通常需要用多重循环来实现，循环次数取决于变量个数，对每个变量的每个值都测试是否满足所给定的条件，如果是则找到了问题的一个解。

递归法

3.7.4 递归法

递归是设计和描述算法的一种有力的工具，由于它在复杂算法的描述中被经常采用，能采用递归描述的算法通常有这样的特征：为求解规模为n的问题，设法将它分解成规模较小的问题，然后从这些小问题的解方便地构造出大问题的解，并且这些规模较小的问题也能采用同样的分解和综合方法，分解成规模更小的问题，并从这些更小问题的解构造出规模较大问题的解。特别是当规模n=1时，能直接得解。

递归算法包括"递推"和"回归"两部分。递推就是为得到问题的解，将它推到比原问题简单的问题的求解。如f（n）=n!,为计算f（n），将它推到f（n-1），即f（n）=nf（n-1），这就是说，为计算f（n），将问题推到计算f（n-1），而计算f（n-1）比计算f（n）简单，因为f（n-1）比f（n）更接近于已知解0!=1.

使用递推时应注意以下条件。

（1）递推应有终止的时候。例如n!,当n=0时，0!=1为递推的终止条件。所谓"终止条件"就是在此条件下问题的解是明确的，缺少终止条件便会使算法失效。

（2）"简单问题"表示离递推终止条件更为接近的问题。简单问题与原问题解的算法是一致的，其差别主要反映在参数上。如，f（n-1）与f（n）其参数相差1.参数变化，使问题递推到有明确解的问题。

回归是指当"简单问题"得到解后，回归到原问题的解上来。如，当计算完f（n-1）后，回归计算nf（n-1），即得n!的值。

使用回归应注意以下问题。

（1）递归到原问题的解时，算法中所涉及的处理对象应是关于当前问题的，即递归算法所涉及的参数与局部处理对象是有层次的。当解一问题时，有它的一套参数与局部处理对象。当递推进入一"简单问题"时，这套参数与局部对象便隐蔽起来，在解"简单问题"时，又有它自己的一套。但当回归时，原问题的一套参数与局部处理对象又活跃起来了。

（2）有时回归到原问题以得到问题解，回归并不引起其他动作。

图的深度优先搜索、二叉树的前序、中序和后序遍历等可采用递归实现。

上一节　本书简介　下一节

作者：希赛教育软考学院　来源：希赛网　2014年05月20日

分治法

3.7.5 分治法

对于一个规模为n的问题，若该问题可以容易地解决（比如说规模n较小）则直接解决；否则将其分解为k个规模较小的子问题，这些子问题互相独立且与原问题形式相同，递归地解这些子问题，然后将各子问题的解合并得到原问题的解。这种算法设计策略叫做分治法。

我们知道：任何一个可以用计算机求解的问题所需的计算时间都与其规模有关。问题的规模越小，越容易直接求解，解题所需的计算时间也越少。例如：对于n个元素的排序问题，当n=1时，不需任何计算。n=2时，只要作一次比较即可排好序。n=3时只要做3次比较即可。而当n较大时，问题就不那么容易处理了。要想直接解决一个规模较大的问题，有时是相当困难的。而分治法的设计思想是将一个难以直接解决的大问题，分割成一些规模较小的相同问题，以便各个击破，分而治之。

如果原问题可分割成k个子问题，$1<k\leq n$,且这些子问题都可解，并可利用这些子问题的解求出原问题的解，那么这种分治法就是可行的。由分治法产生的子问题往往是原问题的较小模式，这就为使用递归技术提供了方便。在这种情况下，反复应用分治手段，可以使子问题与原问题类型一致而其规模却不断缩小，最终使子问题缩小到很容易直接求出其解。这自然导致递归过程的产生。因此，可以说分治与递归像一对孪生兄弟，经常同时应用在算法设计之中，并由此产生许多高效算法。

分治法所能解决的问题一般具有以下几个特征：

（1）该问题的规模缩小到一定的程度就可以容易地解决。

（2）该问题可以分解为若干个规模较小的相同问题，即该问题具有最优子结构性质。

（3）利用该问题分解出的子问题的解可以合并为该问题的解。

（4）该问题所分解出的各个子问题是相互独立的，即子问题之间不包含公共的子问题。

上述的第一条特征是绝大多数问题都可以满足的，因为问题的计算复杂性一般是随着问题规模的增加而增加的。第二条特征是应用分治法的前提，它也是大多数问题可以满足的，此特征反映了

递归思想的应用。第三条特征是关键，能否利用分治法完全取决于问题是否具有第三条特征，如果具备了第一条和第二条特征，而不具备第三条特征，则可以考虑贪心法或动态规划法。第四条特征涉及分治法的效率，如果各子问题是不独立的，则分治法要做许多不必要的工作，重复地解公共的子问题。

分治法在每一层递归上都有以下3个步骤：

（1）分解：将原问题分解为若干个规模较小、相互独立、与原问题形式相同的子问题。

（2）解决：若子问题规模较小而容易被解决则直接解，否则递归地解各个子问题。

（3）合并：将各个子问题的解合并为原问题的解。

二分法查找和汉诺塔问题的求解，就是分治法的典型应用。

第 3 章：数据结构与算法　作者：希赛教育软考学院　来源：希赛网　2014年05月20日

动态规划法

第 3 章：数据结构与算法　作者：希赛教育软考

动态规划法

层一层地分解成一级一级、规模逐步缩小的子问题，直到子问题规模小到可以直接求出其解的子问题为止。分解后的所有子问题按层次关系构成一棵子问题树，树根是原问题。原问题的解依赖于子问题树中所有子问题的解。

与分治法不同的是动态规划法所针对的问题有一个显著的特征，即它所对应的子问题树中的子问题呈现大量的重复。因此，动态规划法的相应特征是对于重复出现的子问题，只在第一次遇到时加以求解，并把答案保存起来，让以后再遇到时直接引用，不必重新求解。

设原问题的规模为n,当子问题树中的子问题总数是n的超多项式函数，而不同的子问题数只是n的多项式函数时，动态规划法显得特别有意义。

动态规划算法通常用于求一个问题在某种意义下的最优解。设计一个动态规划法，可以按照以下几个步骤进行：

分析最优解的性质，并刻划其结构特征。

递归地定义最优值。

以自底向上的方式计算出最优值。

根据计算最优值时得到的信息，构造一个最优解。

步骤~是动态规划算法的基本步骤。在只需要求出最优值的情形，步骤可以省略。若需要求出问题的一个最优解，则必须执行步骤。此时，在步骤中计算最优值时，通常需要记录更多的信息，以便在步骤中，根据所记录的信息，快速地构造出一个最优解。

动态规划算法的有效性依赖于问题本身所具有的2个重要性质：最优子结构性质和重叠子问题性质。

（1）最优子结构。设计动态规划算法的第步通常是要刻划最优解的结构。当问题的最优解包含了其子问题的最优解时，称该问题具有最优子结构性质。问题的最优子结构性质提供了该问题可用动态规划算法求解的重要线索，使我们能够以自底向上的方式递归地从子问题的最优解构造出整个

问题的最优解。

（2）重叠子问题。在用递归算法自顶向下解问题时，每次产生的子问题并不总是新问题，有些子问题被反复计算多次，这就是重叠子问题性质。动态规划算法正是利用了这种子问题的重叠性质，对每一个子问题只解一次，而后将其解保存在一个表格中，当再次需要解此问题时，只是简单地利用常数时间查看一下结果。通常，不同的子问题的个数随输入问题的大小呈多项式增长。因此，用动态规划算法通常只需要多项式时间，从而获得较高的解题效率。

动态规划算法的一个变形是备忘录方法。与动态规划算法不同的是备忘录方法采用的是自顶向下的递归方式，而动态规划算法使用的是自底向上的非递归方式。

上一节　本书简介　下一节

回溯法

3.7.7 回溯法

回溯法是一种选优搜索法，按选优条件向前搜索，以达到目标。但当搜索到某一步时，发现原先选择并不优或达不到目标，就退回一步重新选择。这种走不通就退回再走的技术就是回溯法，而满足回溯条件的某个状态的点称为"回溯点".

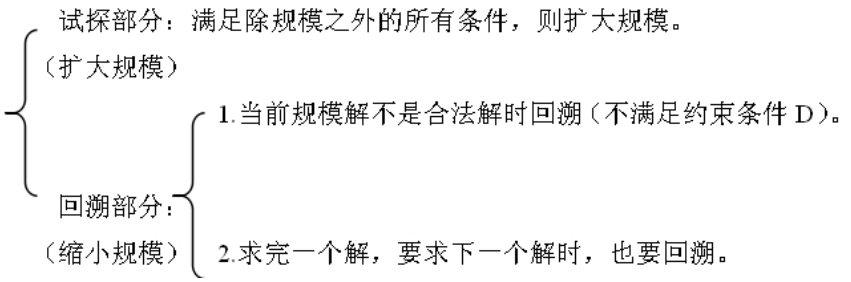
可用回溯法求解的问题P,通常要能表达：对于已知的由n元组（$x_1,x_2,\dots,x_n$）组成的一个解空间$E=\{(x_1,x_2,\dots,x_n)\mid x_i\in S, i=1,2,\dots,n\}$,给定关于n元组中分量的一个约束集D,问题P需要求出E中满足D的所有n元组，其中S是分量x_i的定义域，且 |S| 有限，i=1,2,…，n.称E中满足D的任一n元组为问题P的一个解。

解问题P的最朴素的方法就是穷举法，即对E中的所有n元组逐一地检测其是否满足D的全部约束，若满足，则为问题P的一个解，但显然，其计算量是相当大的。

对于许多问题，只要存在$0\le j\le n-1$,使得（$x_1,x_2,\dots,x_j$）违反D的约束，则以（$x_1,x_2,\dots,x_j$）为前缀的任何n元组（$x_1,x_2,\dots,x_j,x_{j+1},\dots,x_n$）一定也违反D的约束，$n\ge i>j$.因此，可以肯定，一旦检测断定某个j元组（$x_1,x_2,\dots,x_j$）违反D的约束，就可以肯定，以（$x_1,x_2,\dots,x_j$）为前缀的任何n元组（$x_1,x_2,\dots,x_j,x_{j+1},\dots,x_n$）都不会是问题P的解，因而就不必去搜索、检测它们。回溯法正是针对这类问题，利用这类问题的上述性质而提出来的比穷举法效率更高的算法。

回溯法首先将问题P的n元组的解空间E表示成一棵高为n的带权有序树T（称为解空间树），把在E中求问题P的所有解转化为在T中搜索问题P的所有解。例1说明了T的建立和利用T求解的过程。

对于回溯法，我们要搞清楚回溯的条件规则。该算法由两部分组成，如下所示：



上一节　本书简介　下一节

贪婪法

3.7.8 贪婪法

贪婪法是一种重要的算法设计技术，它总是做出在当前来说是最好的选择，而并不从整体上加以考虑，它所做的每步选择只是当前步骤的局部最优选择，但从整体来说不一定是最优的选择。由于它不必为了寻找最优解而穷尽所有可能解，因此其耗费时间少，一般可以快速得到满意的解。当然，我们也希望贪婪算法所得到的最终解是整体的最优解。

贪婪算法的基本思想是在贪婪算法中采用逐步构造最优解的方法。在每个阶段，都做出一个看上去最优的决策（在一定的贪婪标准下），决策一旦做出，就不可再更改。做出贪婪决策的依据称为贪婪准则。

对贪婪法的理解如下：

（1）贪婪法不追求最优解，只求可行解，通俗点讲就是不求最好，只求可好。

（2）每一步都按贪婪准则找一个解，故到n步后（n为问题的规模）得到问题的所有解。如找不到所有解，则修改贪婪准则，放宽贪婪条件或修改算法某些细节，重新从头开始找解。

（3）每一步所找到的解是这一步中的最优解（按贪婪准则来说），但每步最优解所形成的整体解并不一定最优。

应用贪婪法求解问题，首要的问题就是要弄清楚贪婪准则。但是我们应该看到，虽然在每个阶段做出的都是看上去最优的决策，但这些决策的集合从整体来说有可能并不是最优的。

贪婪法是一种不求最优解，只是希望得到满意解的方法，即不求最好，只求可好，求第一次满足条件的解。一般来说，这个解却是最优解很好的近似解。当然，贪婪法所求得的解也有可能是最优解，而且对范围相当广的许多问题它能产生整体最优解。

上一节　本书简介　下一节

分支限界法

3.7.9 分支限界法

分支限界法也称为限界剪枝法，这种算法类似于回溯法，也是一种在问题的状态空间树T上搜索解的算法。但是，分支限界法与回溯法有不同的求解目标。回溯法的求解目标是找出T中的所有回答结点或任一回答结点，而分支限界法的求解目标则是找出T中使得某一目标函数达到极小或极大的一个回答结点，即问题在某种意义下的最优解。

由于求解目标不同，导致分支限界法与回溯法在算法上也有两点不同：

（1）回溯法只通过约束条件，而分支限界法不仅通过约束条件，而且通过目标函数的限界来减少无效搜索，提高求解效率。

（2）回溯法在T上搜索，激活状态结点和选定扩展结点，采用的是深度优先策略，而分支限界法采用的是先广后深的策略，即先检测扩展结点的每一个儿子结点，并将满足约束条件且不越过目标函数的当前限界者激活，使之成为活结点，加入到当前的活结点表中，然后在活结点表中确定下一个扩展结点再继续搜索。下一个扩展点的确定原则体现在对活结点表的组织方式即数据结构之中。活结点表的组织方式有队列式、栈式和优先队列式三种。与这三种组织方式相对应，确定下一个扩展点的原则分别是先进先出、后进先出，以及按活结点的优先级高的先出。

分支限界法的一个典型应用就是求图的最短路径问题。

上一节　本书简介　下一节

概率算法

3.7.10 概率算法

概率算法的一个基本特征是对所求解问题的同一实例用同一概率算法求解两次可能得到完全不同的效果。这两次求解问题所需的时间甚至所得到的结果可能会有相当大的差别。（从此特征，我们就可以排除哈夫曼编码算法，因为哈夫曼编码是一种确定的方法）一般情况下，可将概率算法大致分为四类：数值概率算法，蒙特卡罗（Monte Carlo）算法，拉斯维加斯（Las VeGas）算法和舍伍德（Sherwood）算法。

数值概率算法常用于数值问题的求解。这类算法所得到的往往是近似解。而且近似解的精度随计算时间的增加不断提高。在许多情况下，要计算出问题的精确解是不可能或没有必要的，因此用数值概率算法可得到相当满意的解。

舍伍德算法总能求得问题的一个解，且所求得的解总是正确的。当一个确定性算法在最坏情况下的计算复杂性与其在平均情况下的计算复杂性有较大差别时，可以在这个确定算法中引入随机性将它改造成一个舍伍德算法，消除或减少问题的好坏实例间的这种差别。舍伍德算法精髓不是避免算法的最坏情况行为，而是设法消除这种最坏行为与特定实例之间的关联性。

拉斯维加斯算法不会得到不正确的解，一旦用拉斯维加斯算法找到一个解，那么这个解肯定是正确的。但是有时候用拉斯维加斯算法可能找不到解。与蒙特卡罗算法类似。拉斯维加斯算法得到正确解的概率随着它用的计算时间的增加而提高。对于所求解问题的任一实例，用同一拉斯维加斯

算法反复对该实例求解足够多次，可使求解失效的概率任意小。

蒙特卡罗算法用于求问题的准确解。对于许多问题来说，近似解毫无意义。例如：一个判定问题其解为"是"或"否",二者必居其一，不存在任何近似解答。又如，我们要求一个整数的因子时所给出的解答必须是准确的，一个整数的近似因子没有任何意义。用蒙特卡罗算法能求得问题的一个解，但这个解未必是正确的。求得正确解的概率依赖于算法所用的时间。算法所用的时间越多，得到正确解的概率就越高。蒙特卡罗算法的主要缺点就在于此。一般情况下，无法有效地判断得到的解是否肯定正确。

考点分析

第4章　操作系统

操作系统是软件设计师的一个必考知识点，每次考试的分数在7分左右，主要涉及操作系统的5大管理功能，即进程管理、存储管理、文件管理、作业管理和设备管理。

4.1　考点分析

本节把历次考试中操作系统方面的试题进行汇总，得出本章的考点，如表4-1所示。

表4-1 操作系统试题知识点分布

考试时间	分数	考查知识点
10.11	5	虚拟存储（2）、UNIX的 Shell 程序（1）、PV 操作（1）、银行家算法（1）
11.05	4	虚拟存储器（1）、页式存储（1）、进程调度（2）
11.11	5	树形文件目录（3）、PV 操作（2）
12.05	4	进程调度（2）、UNIX设备管理（1）、文件管理（1）
12.11	5	进程调度（4）、存储管理（1）
13.05	7	进程状态（2）、虚拟设备（1）、LRU（2）、响应时间与吞吐量（1）、位示图（1）
13.11	7	响应时间和作业吞吐量（1）、银行家算法（2）、页式存储管理（1）、多级目录结构（1）、设备驱动程序（2）
14.05	7	PV 操作（2）、文件系统（2）、缺页中断（2）、段式存储（1）

根据表4-1,我们可以得出操作系统的考点主要有以下几个方面：

（1）存储管理：主要考查虚拟存储器，特别是页式存储管理。

（2）进程管理：包括进程状态、PV操作、银行家算法和死锁问题等。在下午考试中，也出现过1次考查PV操作的试题，占15分。

（3）文件管理：主要考查文件的组织和结构、位示图等。

（4）作业管理：包括作业状态及转换、作业调度算法、响应时间和吞吐量等。

（5）设备管理：包括UNIX设备管理、设备驱动程序和虚拟设备等。

（6）Shell程序：主要考查UNIX的Shell程序。

对这些知识点进行归类，然后按照重要程度进行排列，如表4-2所示。其中的星号（*）代表知识点的重要程度，星号越多，表示越重要。

在本章的后续内容中，我们将对这些知识点进行逐个讲解。

表4-2 操作系统各知识点重要程度